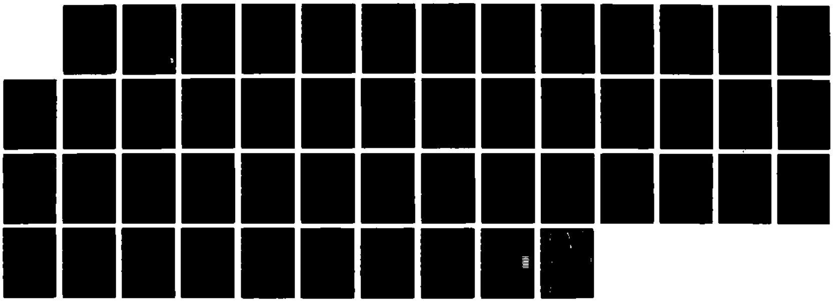
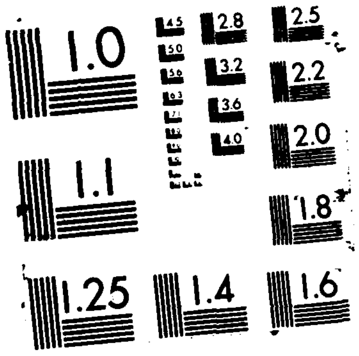
AD-A185 709 GENIE INFERENCE ENGINE RULE WRITER'S GUIDE(U) ARMY BALLISTIC RESEARCH LAB ABERDEEN PROVING GROUND MD F S BRUNDICK AUG 87 BRL-TR-2832 1/1

UNCLASSIFIED F/G 12/5 NL




1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

2

AD

TECHNICAL REPORT BRL-TR-2832

GENIE INFERENCE ENGINE RULE WRITER'S GUIDE

FREDERICK S. BRUNDICK

AUGUST 1987

US ARMY BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A185 709

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) ARBRL-TR-2832	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION USA Ballistic Rsch Lab Vuln/Lethality Div., (LTTB) — 6b OFFICE SYMBOL (If applicable) SLCBR-VL-L	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Aberdeen Proving Ground, MD 21005-5066	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION — 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO — PROJECT NO — TASK NO — WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
GENIE Inference Engine Rule Writer's Guide

12 PERSONAL AUTHOR(S)
Frederick S. Brundick

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Final	FROM Jun 86 TO Apr 87	10 July 1987	59

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Expert Systems; Inference Engine; Artificial Intelligence; Knowledge Representation

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

An expert system is a program that mimics the performance of a human expert in some intellectual endeavor. The inference engine is the heart of the expert system. Genie (GENeric Inference Engine) is a rule-based expert system shell that was initially developed as the basis of an expert system to assist human experts in assessing the vulnerability of turbine jet engines. It is also being used to guide statisticians in performing non-parametric analyses. The program is best suited for diagnostic and classification problems.

The report is intended for the person creating a knowledge base to be used by Genie. A sample knowledge base will be built to show how Genie's various keywords work and how production rules are written. Detailed definitions and restrictions of the keywords, two sample knowledge bases, and sample runs are presented in the appendices. A glossary is included to explain the terms used.

(Continued)

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL FREDERICK S. BRUNDICK	22b TELEPHONE (Include Area Code) 301-278-3091 — 22c OFFICE SYMBOL SLCBR-VL-L

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. (continued)

—> An inference engine shell will aid the development of future expert systems, while the expert systems themselves will make the vulnerability analyst's job much easier. This guide will enable anyone to create his own expert system.

/)

TABLE OF CONTENTS

I. INTRODUCTION

An expert system is a program that mimics the performance of a human expert in some intellectual endeavor. A basic tenet of the expert system methodology requires separation of domain expertise from the strategies for manipulating and applying that expertise; this modularity greatly facilitates debugging and modifying either subsystem. Figure 1 shows a typical expert system and a few of the major subsystems of the inference engine.

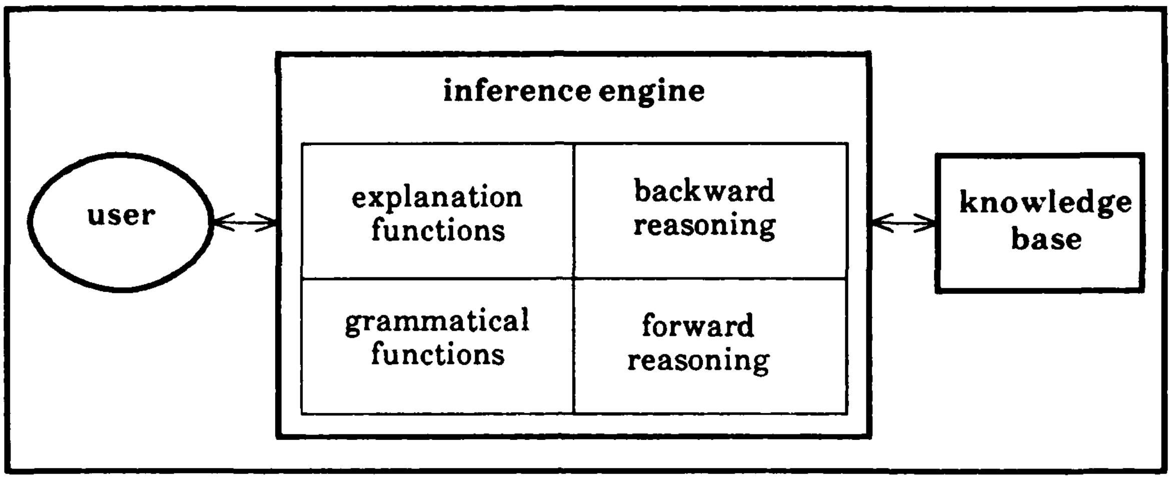


FIGURE 1. *Typical Expert System*

The inference engine is the heart of the expert system. It interprets the production rules and other data structures stored in the knowledge base (KB), queries the user for input, and draws conclusions. Genie (GENeric Inference Engine) is a rule-based expert system shell that was initially developed as the basis of an expert system to assist human experts in assessing the vulnerability of turbine jet engines.* It employs two control strategies, called *backward* chaining (or goal driven) and *forward* chaining (or data driven). In backward chaining Genie selects one of the KB's hypotheses and attempts to accumulate supporting evidence to verify the statement, while in forward chaining it applies known facts to the production rules to learn new facts.

Ease of use was a fundamental objective in developing Genie. The rule writer (also called the knowledge engineer) should not have to learn a new computer language, but should be able to concentrate on building his KB. Genie's facts are therefore written in everyday English with only a few restrictions. While Genie never actually "understands" the meaning of a statement, the grammatical functions can convert a statement from affirmative to negative and vice versa or cast the statement in the form of a question.

The ability of the expert system to explain its actions is another desirable feature. The user can ask Genie why it is asking him a particular question, what the current goal (hypothesis) is, and what facts have been learned thus far. Genie also keeps an audit trail so that the user can see how it reached its final conclusion. This serves both to point out errors in the knowledge base and to assure the user that the solution is correct.

*Brundick, F.S., et al. *GENIE: An Inference Engine with Applications to Vulnerability Analysis*. BRL Report TR-2739, June 1986.

This report is intended for the person creating a knowledge base to be used by Genie. A sample KB is built to show how Genie's various keywords work, and new terms are explained as they are encountered. Detailed definitions and restrictions of the keywords are presented in the appendices for reference. The example is intentionally kept simple to demonstrate Genie's underlying concepts and is not meant to be a guide for knowledge engineering.

There are three implementations of Genie available -- one in Franz LISP running under UNIX, a recently completed Common LISP translation for Gould UTX/32 2.0, and a larger INTERLISP-D version for Xerox 1108 and 1186 LISP Machines. The INTERLISP-D version makes use of multiple windows, pop-up menus, different fonts, graphics, mice, and other features unique to the Xerox environment. These differences will not affect the rule writer, since the same KB may be run on all three systems with no changes. The sample runs in Appendices E and G were generated with the Franz version.

II. GETTING STARTED

Genie is best suited for classification and diagnosis problems; our example involves building a simple expert system to identify various animals. The first step is to determine what animals the system should "know" about. this KB will use bat, giraffe, zebra, tiger, cheetah, lion, ostrich, penguin, robin, owl, vulture, goldfish, shark, and piranha. Next the animals are divided into categories One natural grouping is mammal, bird, and fish, while two subsets of mammals are ungulates and felines. Partitioning the data into distinct groups speeds up the inference process and eliminates many superfluous questions. For example, if Genie has determined that the animal is a mammal then it can ignore all birds and fish, and it will not ask irrelevant questions like "Does animal have feathers?". (Of course, if Genie incorrectly assumes the animal is a mammal then it will not reach the correct conclusion This problem is not restricted to the use of groups and is another point of departure between traditional and AI programming.)

The groups and their members are graphically shown in Figure 2. Notice that this tree is much simpler than the trees in a "real" system; this is partly because the knowledge base is small and easily partitioned into distinct groups

The links in the trees represent rules, where the top fact is one of the antecedents and the bottom fact is the conclusion. For example, the mammal-ungulate link means "If the animal is a mammal and certain other conditions are met then the animal is an ungulate", while the ungulate-zebra link means "If the animal is an ungulate and the other conditions are true then the animal is a zebra". The hypotheses (individual animals) are called *leaf* nodes or simply *leaves*. (A second type of leaf node will be discussed later.) The boxed nodes (groups) are *intermediate* nodes. The topmost node in the tree (ANIMAL) is called the *root* node, although Genie does not use an explicit root node; the term is mentioned for completeness.

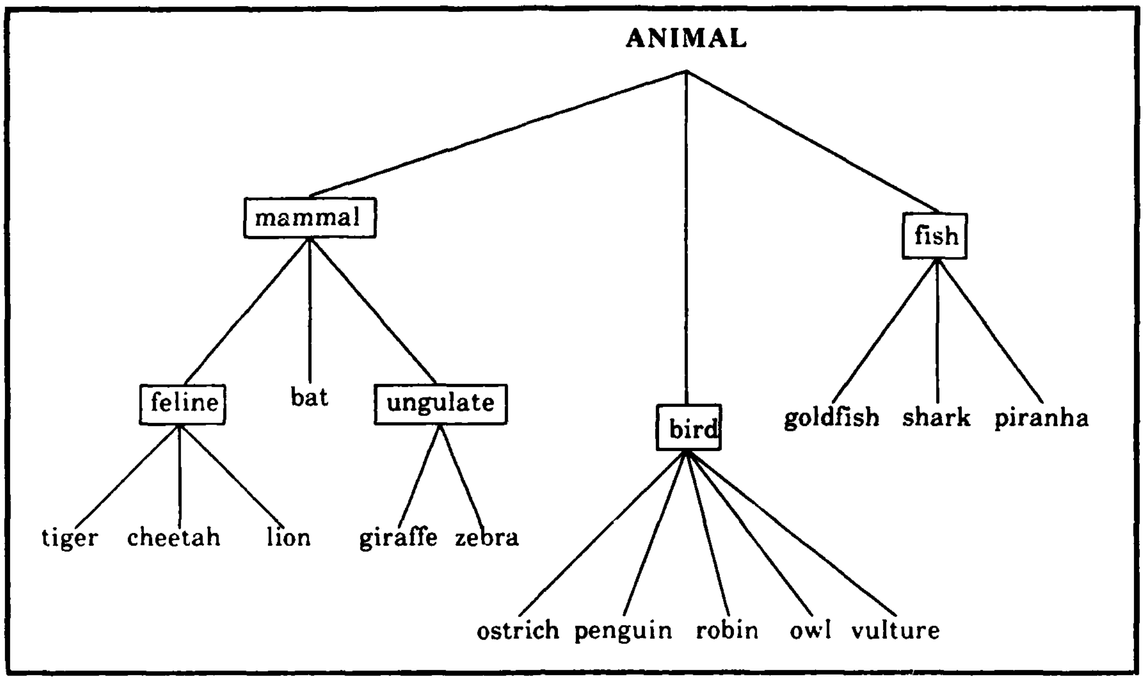


FIGURE 2. *Animal Groups*

Once you've chosen your initial groups (if any) you can begin writing rules, so let's start with the fish rule. Three conditions are adequate to describe a fish as shown in the *production* (or if/then) rule below:

```
IF     animal has scales
 and   animal has fins
 and   animal swims
THEN   animal is fish
```

There are two different rules about mammals instead of just one:

```
IF     animal has hair
 and   number of legs LE 4
THEN   animal is mammal

IF     animal gives milk
THEN   animal is mammal
```

The two sets of facts that define "birdness" are:

```
IF     animal has feathers
THEN   animal is bird
```

```
IF      animal flies
 and    animal lays eggs
 and    number of legs EQ 2
THEN    animal is bird
```

Rule writing is an iterative process of testing and refining rules until the system performs correctly. Since the Animal knowledge base is presented as a finished product, here is an explanation of these rules in some detail. The fish rule says the animal must possess three characteristics to be a fish, although some of the facts seem redundant. If you search the Animal KB you will discover that scales and fins are mentioned only in this rule, so either one would be enough to uniquely identify a fish. But there are animals other than fish that have scales, and if someone were using this system to identify an animal that Genie did not know about (*e.g.*, a garter snake) then he might think Genie had correctly identified the animal when it really had not. These rules were taken from a much larger KB that included reptiles and insects, so certain facts may actually be irrelevant. Notice that milk and feathers were considered to be distinctive enough to classify the animal as a mammal or bird, respectively. On the other hand, while all mammals have hair, not all hairy animals are mammals, so an additional fact involving the number of legs was required so that Genie would not confuse a zebra with a tarantula. Likewise, not all flying, egg-laying animals are birds.

In the sample rules all the antecedents were written with *AND*s, meaning all the facts had to be true in order for the rule to fire and the conclusion (there may be more than one conclusion) to be added to the dynamic portion of the KB. With the exception of an option that will be discussed later, the antecedents are always *AND*ed together, and the rule writer does not actually type the word **and** in the rule. When Genie learns a new fact it forward chains on the fact, testing each rule it appears in to see if the rule will fire. If other facts are deduced they will also start a series of forward chaining tests. Eventually no more rules will fire, all the new facts will have been tested, and the chaining will stop.

In contrast to this, rules which reach the same conclusion are *OR*ed together. When Genie is given a fact to verify, the first thing it does is find out if the fact appears as the conclusion of any rules. If so, it backward chains on the first of these rules and attempts to sequentially verify each fact in that rule. It will recursively climb the tree until it reaches a fact which is a leaf node (*i.e*, a non-intermediate node) and ask the user if the fact is true. If a rule fails Genie will try the next rule that deduces the desired fact, if all the rules fail the fact is stored with a truth value of *indeterminate*.

The tree in Figure 3 shows how Genie would attempt to prove that the animal is a mammal. Genie begins by backward chaining on rule1. The first fact in this rule is "animal has hair" which is a leaf node, so Genie will ask the user if the animal has hair. If so, it will ask how many legs the animal has. If the number given is less than or equal to 4, rule1 will fire and Genie will learn that the animal is a mammal. If the animal does not have hair or it does have hair but the number of legs is greater than 4, Genie will backward chain on rule2 and ask if the animal gives milk. If it does give milk, rule2 will fire and Genie will deduce that the animal is a mammal. If it does not give milk, then Genie will mark "animal is mammal" as *indeterminate*, meaning it tried to determine the truth of the statement but failed (see *DEFAULT* below). Regardless of the outcome (*true* or *indeterminate*) Genie will forward chain on "animal is mammal". Similar trees could be drawn for the bird and fish rules

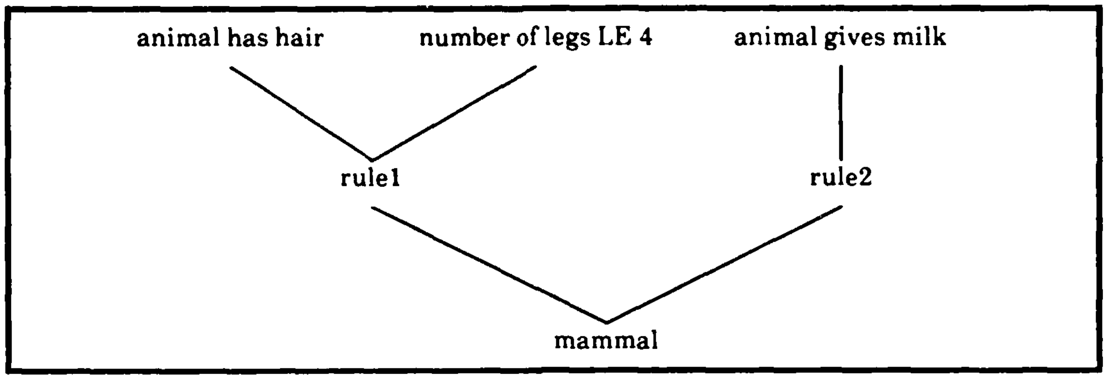


FIGURE 3. *Mammal Subtree*

The animals may be divided into two other groups, namely carnivores and non-carnivores. There are two rules that reach the conclusion "animal is carnivore". They are:

```
IF      animal eats mostly meat
THEN    animal is carnivore

IF      animal has pointed teeth
 or     animal has claws
 or     animal has forward-pointing eyes
NEED    1
THEN    animal is carnivore
```

The second rule has mammal-specific features which means it is not quite correct (how many carnivorous birds have pointed teeth?), but that's not the point of this rule. This production rule has a *NEED 1* clause, which turns it into an *OR* rule. It would have been perfectly correct to have written the second rule as a series of three simple rules where each contained only a single antecedent. One reason for allowing the NEED clause was to minimize the number of rules in a KB; it is also easier to understand since several trivial rules have been grouped together. The original reason for the NEED option was to create rules where most of the facts (but not all) were needed, as in medical diagnosis. A patient may not have all the symptoms that would indicate a particular disease, so the doctor would use a heuristic rule such as "If the patient has 'most' of the symptoms, then he has the disease". Such a rule would be written as:

```
IF      patient has symptom 1
        patient has symptom 2
        patient has symptom 3
        patient has symptom 4
        patient has symptom 5
NEED    3
THEN    patient has disease X
```

This means the rule will fire if ANY three of the file antecedents are true. A more sophisticated version of a NEED rule acts as a combination *AND/OR* production rule as shown in the example below.

```
IF      animal is ungulate (10)
        markings are dark spots (10)
        animal has long neck (1)
        animal has long legs (1)
NEED    21
THEN    animal is giraffe
```

The numbers in parentheses are not weights in the normal sense, so "animal is ungulate" is *not* 10 times as important as "animal has long neck". If you code the facts as A, B, C, D, and E then this rule is saying "the animal is a giraffe if both A and B are true and one (or both) of C or D are true"; in a more precise form, E = A and B and (C or D). Genie will know this rule has failed if the animal is not an ungulate or if it does not have dark spots, regardless of the truth of the other three facts, because both A and B must be true for the true facts to total 21 There is nothing special about the values 10 and 1, but you must be sure that the rule will not fire on an unexpected combination of facts. Changing the "weights" of A and B to 2 and the needed total to 2 will mean E = A or B or (C and D). It is impossible to choose a set of numbers which would mean E = (A and B) or (C and D); in that case you must write two rules. This was done intentionally to prevent the rule writer from writing complicated rules that would be difficult to understand and debug

III DEFINITION OF TERMS

The terms *fact* and *truth* have been used without being formally defined In a Genie KB, each piece of information is called a *fact statement* or simply a *fact*. The grammatical portion of Genie can perform primitive operations on facts, such as converting from affirmative to negative and vice versa and rewriting a fact in the form of a question. Every fact in the KB has an initial truth value of *unknown*, with the other possible values being *true*, *false*, and *indeterminate*. A fact's truth may become *true* or *false* when the user answers a question or makes a menu selection, a numeric fact has values for all of its component variables, an exclusive-or set is evaluated (see *XOR* below), or a rule fires. All of the examples so far have involved only affirmative facts, so here are a couple of negative fact examples:

```
IF      animal is bird
 and    animal does not fly
 and    animal is black and white
 and    animal swims
THEN    animal is penguin

IF      animal eats meat
THEN    animal is not vegetarian
```

The first of these rules contains a negative antecedent: animal does not fly. Genie will store this fact (as it stores all facts) in its affirmative form, *i.e.*, animal flies, and record a desired truth value of *false* in the rule. When Genie tests the rule to see if it can fire, it will compare the desired truth of each fact with the fact's known truth In this case, if the fact is false then the rule will be satisfied.

To avoid confusion between known truth and desired truth, Genie uses the terms *true* and *false* with the former and *succeeded* and *failed* with the latter; see rule3 and rule18 in Appendix E. The second rule has a negative conclusion; if the animal does eat meat then the single antecedent will succeed, the rule will fire, and Genie will record the truth of the statement "animal is vegetarian" as *false*.

While *true* and *false* are familiar through everyday usage, the term *indeterminate* requires some explanation. It is not the same as *unknown*. If Genie needs to know a fact's truth value, and the value is currently unknown, then Genie will backward chain or ask the user a question to determine the truth value. If the backward chaining fails or the user does not know the answer, then the fact becomes indeterminate. This is Genie's way of saying "I tried to learn the fact's truth but I couldn't, so I'll forget about it and throw out all the rules that use this fact (with the possible exception of *NEED* clauses)". Rules always require antecedents to have values of *true* or *false*, so if a fact is *indeterminate* then it must always fail.

Sometimes you will want to override an indeterminate fact; for instance, you might decide that all mammals must adhere to one of the two mammal rules discussed earlier. You can tell Genie (via the keyword *DEFAULT*) to record the truth value of "animal is mammal" as *false* if both rules fail. The *DEFAULT* clause may be thought of as a meta-rule that allows the rule writer to say "If a fact is indeterminate, then assume the following truth value". This is the only way that the rule writer may refer to an indeterminate fact. In addition, default values are only used with intermediate facts and are ignored by facts that the user is asked about directly. When you actually run the inference engine and a default value is used, the fact will be displayed after the word **Assumption**, and if you ask Genie how it learned that fact it will tell you it used a default value because all the rules leading to that conclusion failed.

IV. WRITING PRODUCTION RULES

There are a few restrictions that you must know about before you can write your own rules. First of all, rules may not be recursive, *i.e.*, if you backward chain on any path through the fact tree you should never see the same fact more than once. The following rules are directly recursive:

```
IF     animal eats mostly meat
THEN   animal is carnivore

IF     animal is carnivore
THEN   animal eats mostly meat
```

Here is a set of rules that are indirectly recursive:

```
IF a THEN b
IF b THEN c
IF c THEN d
IF d THEN a
```

Genie does not check for recursion, so it is very important that you do not create recursive rules; if Genie tries to backward chain on a recursive fact it will hang in a loop. Remember that Genie manipulates all facts in their affirmative form, so "IF a THEN b" and "IF b THEN not a" are recursive.

Suppose that you had several disjoint facts which belonged to the same set, *e.g.*, an animal may be a mammal or a bird or a fish, but it cannot be a mammal-bird or a bird-fish or a mammal-fish. If Genie learned (during a run) that an animal was a fish, then it could ignore all the mammals and birds; it would not bother to ask the user useless questions like "Does the animal have hair?". Using normal production rules, you would have to write:

IF animal is mammal
THEN animal is not bird
and animal is not fish

IF animal is bird
THEN animal is not mammal
and animal is not fish

IF animal is fish
THEN animal is not mammal
and animal is not bird

Not only are these rules tedious, but they are also recursive! Genie uses the keyword *XOR* (exclusive *OR*) to implement the concept of mutually exclusive facts. Instead of the rules given above, you would write:

XOR animal is mammal
animal is bird
animal is fish

Genie treats XORs as a subset of production rules, so if it is trying to prove "animal is mammal", both the mammal rules fail, and there is no default value, then it will backward chain on "animal is bird". This appears to be recursive, but Genie keeps track of the XOR facts that it is testing to prevent loops from forming. XOR "rules" fire only if a fact is *true* (affirmative) as shown in the three expanded rules. If Genie learns that the animal is a mammal, then it knows the animal is not a bird and not a fish, but if it learns the animal is NOT a mammal and the animal is NOT a bird it does not assume that the animal is a fish.

V. MENUS

If you look at the mammal/bird/fish rules given earlier, you will notice that each rule has a fact of the form "animal has X" where X is one of hair, feathers, or scales. These rules will work as written, but it is possible that Genie will ask the user about each of these skin coverings as it searches the tree for a solution. A sample dialog could go:

Does animal have hair? *no*

...

Does animal have feathers? *no*

...

Does animal have scales? *yes*

It would be nice to simply ask the user "What is the animal's skin covering?" and resolve the matter in a single question, but now we run into problems with natural language. If the user says *fur*, Genie must know that is the same as *hair*. There is a finite number of responses that are valid here (hair, feathers, scales, and *unknown*), so Genie just displays all the choices in a menu and lets the user pick one. There should be a special entry for "none of the above" to handle peculiar animals like whales and for animals that this KB does not know about like grasshoppers. All facts in a menu are automatically put into an XOR set on the assumption that only one of the facts may be true.

While the exact syntax for *MENU* is given in Appendix A, here is an explanation of the details. Each menu has a single *template* and one or more *fragments* or *choices*. When Genie compiles the KB, it substitutes each fragment into the template and generates a list of fact statements which you must use in the KB. The template looks like any other fact, and you put an **X** where each fragment should be inserted; the **X** may appear anywhere in the template but is normally placed at the end. For example, here is how the skin covering menu would be entered:

```
MENU
    animal has X
        (hair)
        (feathers)
        (scales)
```

Each fragment has parentheses around it to allow for multiple words. If you wanted to refer to these facts in rules, you would use:

```
animal has hair
animal has feathers
animal has scales
```

When Genie builds a menu, it displays a "question", the choices, and a prompt. The question is based on the template unless the rule writer has supplied a special question. The example above would use the default question of "Animal has:", whereas the template "Animal has **X** fur" would generate the question "Animal has --- fur". In the Animal KB, the statement "Animal has:" was too vague, so the MENU clause uses the question "What is the skin covering?". Likewise, the template "markings are **X**" does not say much, so its menu asks "What kind of markings does animal have?".

VI. NUMERIC FACTS

The facts discussed so far have been logical (true/false) facts, *e.g.*, animal flies or animal does not fly, or logical facts that have been grouped together in a menu. Real data also include numbers, which Genie manipulates in numeric facts. The sample rules presented earlier show the simplest form of a numeric fact: Variable Relation Number. Variable has the form "Attribute **of** Concept" (usually referred to as AofC), Relation is one of the 6 numeric inequalities (*e.g.*, GT, >, LE, <=, etc.), and Number is a real or integer constant. A *concept* is an item or object and an *attribute* is some property of the item. Some examples of attribute/concept pairs are weight of person, number of legs, and time of day.

Each variable may be thought of as a sub-fact and is manipulated almost like a fact in a rule. When Genie encounters a numeric fact (*e.g.*, number of legs LE 4) it looks in the dynamic portion of the knowledge base to see if the fact's variable has been assigned a value. If not, it asks the user for a value, and if the user does not know a value it backward chains on the variable. Instead of using production rules, you supply the KB with an equation to evaluate to generate an estimate of the number. In principle, this idea is similar to the *if-needed* daemons used in some frame-based systems. The equations are LISP expressions with the list "(Attr of Concept)" substituted for each independent variable. If the user does not know the value of an independent variable, Genie will backward chain on the variable if it has a default equation.

Numeric facts may only appear as the antecedents of production rules. The rule writer may "customize" a variable by using the *NUMBER* keyword. He may supply a default equation, the allowable range of values (for input checking), the number of decimal places to use when displaying the value, and the data units for the variable (see Appendix A). Genie does not know what the data units mean; they are displayed with the value of the variable and when the user is asked to supply a value. No type checking or conversion of input values is performed at this time.

While only the simplest form of a numeric fact was used in the Animal KB, there are actually several forms which may be used. The general form is "Left-Hand-Side Relation Right-Hand-Side" (or "Lhs Relat Rhs") where each side of the equation may be a simple constant, an AofC, or a LISP expression that evaluates to a number. If you use an expression, each AofC must have parentheses around it, while for simple AofCs the parentheses are optional. A bounded numeric statement has the form "Number Relation AofC Relation Number" where the relations must be LT or LE. Rather than belabor the point, here are some examples which show valid numeric facts

```
number of legs LE 4
height of john > height of lynn
(quotient (length of item) (diameter of item)) GT 2.5
3 <= number of people < 6
```

These facts mean

Is the number of legs less than or equal to 4 ?
Is John taller than Lynn ?
Is the item's length divided by its diameter greater than 2.5 ?
Are there 3, 4, or 5 people ?

The Animal KB contains only a couple of simple arithmetic facts; refer to Appendix F for a demonstration numeric knowledge base that uses several variables, complex numeric facts, and default equations. Appendix G shows how units and bounds are used in a sample run.

VII. OTHER OPTIONS

Earlier it was mentioned that Genie generates its own questions for logical facts, variables, and menus. Menu questions are entered along with the menu template and choices within the *MENU* clause. You may supply your own question for non-menu facts and variables with the *QUESTION* keyword. The syntax is simple, but remember that the question is displayed exactly as you write it; a blank is added after the question to separate it from the user's response, but you must supply the question mark or other punctuation and the initial capital letter (if desired). Two simple examples are shown below:

```
QUESTION
     animal is scavenger     Is the animal a scavenger?
```

```
QUESTION
    length of thread        How long is the threaded portion of the
                              item (inches)?
```

The *DEFAULT* keyword was discussed in Section III. A default is a list of two fact statements where the first must be an affirmative statement and the second may be either affirmative or negative, but both must refer to the same fact. Unlike the other keywords discussed so far, where each keyword is applied to a single object, more than one default pair may be given in a single *DEFAULT* clause. You may prefer to put each default pair in a separate clause for clarity (as shown in Appendix C). Here is an example using Animal data:

```
DEFAULT
    (animal is mammal     animal is not mammal)
    (animal is bird       animal is not bird)
    (animal is fish       animal is not fish)
```

You only need to supply default values for *intermediate* facts (the boxed nodes in Figure 2). If the user does not know if a simple fact is true, Genie records a truth value of *indeterminate*, ignoring any default value that the rule writer may have specified. All hypotheses are implicitly assumed to be *false* because the main objective is to prove what something *is*, not what it is *not*.

The next keyword is in some ways the most important, as Genie must be told what the hypotheses are via *HYP*. When the inference engine is run, it randomly orders all the hypotheses and attempts to verify each of them until one has been proven to be true or all of them have failed (you may set a flag to force Genie to test every hypothesis). The order that the hypotheses are put into the KB is not important, nor does it matter how they are grouped into *HYP* clauses. In the Animal KB multiple *HYP* statements were used with similar animals grouped into each clause:

```
HYP
    animal is bat

HYP
    animal is tiger    animal is cheetah    animal is lion

HYP
    animal is giraffe  animal is zebra
```

A backward chaining expert system tends to ask questions in a logical or "natural" order because it follows the same train of thought that a human expert would use. For instance, if Genie is trying to prove that the animal is a mammal it will ask questions that have a direct bearing on "mammalness", not questions about the animal's markings or what it eats. However, there are times when you may wish to ask certain questions at the very beginning of an identification run: the classic example is asking for the patient's name, age, and sex before attempting to diagnose his illness. While these bits of information may never be needed during the analysis (the name is only required for record-keeping purposes), people are used to giving background information at the start.

The keyword *ASK-FIRST* is used to tell Genie which facts to verify first. Notice that Genie does not simply ask if the fact is true, but actually attempts to prove it by backward chaining. This is how you can override the order of the hypotheses and force a given hypothesis to be tested first. If a fact is usually true, then make that fact an ask-first, likewise a mutually exclusive fact may be proven to prune the search tree and eliminate some of the hypotheses. Like hypotheses, ask-first facts may be grouped into single clauses or scattered in separate clauses, but order *is* important, as the facts are verified (or asked) in the same order that they are encountered in the KB file. Each item in an *ASK-FIRST* clause is either a fact statement or an AofC:

```
ASK-FIRST
    number of legs
    animal is mammal
```

Genie always announces each conclusion it reaches when a rule fires or when a default value is used. This feedback is intended to show the user that his input is being analyzed and the expert system is making inferences and generally working toward a solution. (The alternative is to ask many questions and then give the solution like a magician producing a rabbit.) However, there may be times when you create an intermediate fact to use as an artificial group or internal flag, but do not want the user to be aware of the odd fact. Put the fact in a *NOSHOW* clause and Genie will not report when it learns the fact's truth, but otherwise will treat it like any other fact. This keyword was added for a specific knowledge base and has not been used since then. Numeric facts (*e.g.*, number of legs LE 4) are never displayed when they are learned because a single variable could appear in many facts; of course the user may ask Genie to show all the numeric facts that use a given variable. Multiple facts may appear in a *NOSHOW* clause:

```
NOSHOW
    animal is feline
    show pins menu
```

If Genie tests all the hypotheses and fails to reach a conclusion it displays "No hypothesis can be confirmed." You may replace this message with your own via the *NULL-HYP* keyword:

```
NULL-HYP
    You should visit the zoo more often.
```

VIII. HOW TO WRITE FACT STATEMENTS

There are a few rules to be observed when writing fact statements:

1. A fact statement is a list of words enclosed in parentheses. The parentheses were left off in the earlier examples, so study the Animal KB in Appendix C and the Numeric KB in Appendix F.

2. Genie is case sensitive -- always use lower case. Questions and statements are converted into "proper" sentences before they are displayed (except for questions that the rule writer supplied).

3. Do not use punctuation or parentheses. Commas, quotes, and parentheses have special meanings to LISP; you may use single quotes only in the INTERLISP-D version. Genie's parser does not know that "isn't" is the same as "is not", so avoid problems and spell everything out.

4. Avoid the use of "filler" words like *a*, *an*, *the*, etc. Remember that Genie uses English but does not understand it; sometimes you may be forced to reword a fact to make it more acceptable to Genie. If you do not like the way the grammar package has worded a question, then supply your own with a *QUESTION* clause.

5. Numeric facts must follow a very rigid format. Variables **must** be exactly three words long and be of the form "attribute **of** concept". Convert multiple words to single ones by putting dashes between the words, *e.g.*, number of left-handed-framistans. See Appendices A and F for examples of valid numeric facts.

6. The grammar package is basically a matcher, not a parser. Two facts are the same if and only if they are identical; "car has four doors", "number of doors EQ 4", and "car is a four-door" are three distinct and different facts. Be consistent! Note, however, that numeric facts may use symbolic or alphabetic comparisons interchangeably and you can use whichever forms you are most familiar with. They were intentionally mixed up in the examples.

7. There is only one "correct" negative form of a given affirmative statement. If you are not sure how Genie will process a fact, manually call the inversion function. it is named **complement** in the Franz version and **COMPLEMENT.GENIE** in INTERLISP-D. For example, (complement '(animal has wings)) should return (animal does not have wings). and (complement '(item is not hollow)) will return (item is hollow). To see how Genie will word a question, use the functions **build-question** and **BUILD.QUESTION.GENIE**. If you must negate an "irregular" fact, put the word ***NOT*** at the beginning of the fact. Do not use negatives with numeric facts; instead employ the opposite comparison. In other words, if you want to say "animal does not have more than 4 legs" the correct form is "(number of legs LE 4)".

8. Only use present tense. This should not be a problem unless you refer to past actions, in which case you may need to rewrite the facts or supply a special question.

IX. BUILDING A KNOWLEDGE BASE

Now that all the keywords that Genie understands in a KB have been explained, here are some further details that a rule writer must be aware of. The rule writer puts all the production rules and other clauses into a file using his favorite text editor. Blank lines may be used. A semicolon causes everything following it on the line to be interpreted as a comment (*i.e.*, ignored) by Genie's input routines.

The entire Animal KB is presented in Appendix C, and here are the first few lines:

```
; This is the Animals data as used by Genie version 5.5

; = = = = = = =  If/Then rules  = = = = = = = =

; determine general class [mammal, bird, fish]
(IF (animal has hair)
    (number of legs LE 4)
 THEN (animal is mammal))
(IF (animal gives milk)
 THEN (animal is mammal))
(IF (animal has feathers)
 THEN (animal is bird))
(IF (animal flies)
    (animal lays eggs)
    (number of legs EQ 2)
 THEN (animal is bird))
(IF (animal has scales)
    (animal has fins)
    (animal swims)
 THEN (animal is fish))

; determine subclass [carnivore, ungulate, feline]
(IF (animal eats mostly meat)
 THEN (animal is carnivore))
```

The different types of clauses may be intermingled, but consider separating them both for clarity and to make future modifications easier; *e.g.*, the production rules might be followed by the hypotheses, then the mutually exclusive facts, etc. Remember that order is important in *ASK-FIRST* clauses but not important for the hypotheses because of the way in which they are processed.

When the inference engine is run, it first shuffles all the hypotheses, verifies each of the *ASK-FIRST* items, and then attempts to prove each of the hypotheses, stopping when a solution is found or the hypothesis list is empty. The heart of the engine is the function **verify** (or **VERIFY.GENIE**). When **verify** is given a fact to prove, it checks the KB to see if the fact's truth has already been determined. If not, and if the fact appears as the conclusion of any rules, then the list of rules is passed to the function that performs backward chaining. (If the fact is a leaf node the user is simply asked if the fact is true.)

There are a number of different strategies that expert systems can use to decide which rule should be verified first when backward chaining. One method is to check the rule with the fewest antecedents, on the assumption that the system will run faster; another is to use the rule with the most antecedents because the most information will be learned; and a third is to test the rule with the fewest unknown facts. Likewise, when a fact is being tested, in what order are the antecedents verified? Do you ask the simple facts first, or the fact that appears in the most rules, or just shuffle the facts like the hypotheses?

Genie takes the approach that there is no single best answer, because all knowledge bases are different and what works in one system may be a poor choice in another. Therefore, all rules, and the antecedents within the rules, are processed in order of appearance. If fact5 is the conclusion to rules 1, 2, and 4, and **verify** is backward chaining on fact5, then rule1 will be tested first; if it fails rule2 will be tested next, and finally rule4. If rule4 fails, then fact5 is *indeterminate* (see Section II). This means that you must be careful when you construct your rules. Try putting intermediate facts at the top of a rule to force the search tree to be pruned early; the user will be asked detailed questions only when the selection process has focused on highly likely candidates. For example, here is the shark rule:

```
(IF (animal is fish)
    (animal is carnivore)
    (animal is big)
 THEN (animal is shark))
```

The first two facts refer to major groups that the animal may belong to. If the animal is not a fish or it is not a carnivore (even if it is a fish), then Genie will not bother to ask the user if the animal is big because it cannot possibly be a shark. On the other hand, if the KB does not know yet if the animal is a fish, then Genie will attempt to prove that it is a fish. Because fish, mammal, and bird are mutually exclusive facts, the search will (probably) be narrowed to one of the three groups.

For the sake of efficiency, a rule is pre-tested before any of its antecedents are verified. If a rule has five antecedents, no *NEED* clause, and the fourth fact has already failed, then there is no reason to consider the rule because it cannot fire, even if the other four antecedents all succeed. (Rules with *NEED* clauses are pre-tested as well, but the details are beyond the scope of this report.) The forward chaining function calls the same pre-test routine, so most rules are eliminated before the backward chaining function examines them.

X. PACKAGING THE EXPERT SYSTEM

You should now be able to write your own knowledge base; please study the Animal and Numeric Test KBs in the appendices for additional examples. Genie is a simple programming language, and like any other new language you must practice until you are comfortable with all the features and their syntax, uses, and quirks.

Genie requires some information that is not provided in the knowledge base file. The assumption is being made that the Franz version of Genie is already running on your system, so the operating system and installation details will not be explained here (they are available with the Genie source code). You must create a small LISP file that tells Franz to load and execute the Genie package, the names of various files, and the state of a few flag variables. This file must end with .l (that is an l as in LISP), and you should use an obvious name, *e.g.*, the file that runs the Animal system is called **animals.l** The rest of this section gives a line by line explanation of **animals.l**, which is reproduced in Appendix D:

It is nice to begin by displaying a banner:

```
(msg N "Animal Identification Demo" N N)
```

Next load the Genie system:

```
(load 'genie)
```

The first time Genie reads a KB, it converts all the knowledge into a form it can handle, stores the information in data structures called *frames*, and places this "digested" data into a new file. From then on, when the bootstrap file (in this case, **animals.l**) is loaded Genie will read the digested file and ignore the original file. There is nothing special about the file names; the examples in this report use the suffixes **.english** and **.frames** to emphasize what kind of files they are. Here is how you tell Genie the file names:

```
(setq InputFile    'animals.english)
(setq DigestedFile 'animals.frames)
```

The grammar functions learn about irregular verbs from the variable **VerbPairs** (see below). One verb that was specific to the Animal KB was added :

```
(setq VerbPairs (cons '(flies fly) VerbPairs))
```

This file did not use the two flag variables, whose default values are **nil** (or false). If the variable **TryAllHyps** is non-**nil**, Genie will attempt to verify all the hypotheses and not stop when one is proven. Most of the inference functions have one or more debug statements in them to allow you to see how Genie is testing various facts and rules; this trace may be turned on by setting the variable **Debug** to a non-**nil** value Debug mode was originally implemented to test Genie as it was being developed and is still used to test modifications and extensions to the inference engine. If you run a debug trace on a very small KB and study the results you will get a good idea of how Genie's inference engine works. The user's ability to ask *how* and *why* are not related to debug mode. Here are the statements to turn on both of the flags:

```
(setq TryAllHyps t)
(setq Debug t)
```

The last two steps are to initialize the system and start the command interpreter. The function **initialize** reads the digested file or compiles the input file, while the function **driver** prompts the user for input and processes his commands. You may give **driver** an initial command, and the logical choice is to immediately start the inference engine:

```
(initialize)
(driver go)
```

Genie consists of several files containing compiled LISP functions and one file of variables. This file, **auxvar.l** (for **auxiliary variables**), defines the keywords that Genie recognizes, patterns for different kinds of sentences, and some other things that only the Genie maintainer needs to be concerned with The variable of interest to the rule writer is **VerbPairs** When **complement** or **build-question** manipulates a fact, it needs to find the primary verb in the fact. The auxiliary verbs that are known are *is*, *are*, and *can*. If one of these words is not found, the matcher looks for a verb that appears in **VerbPairs**, and if it still cannot find the verb it uses the first word that ends with an *s*. When a fact contains an irregular verb, both the singular and plural form of the verb must be stored in **VerbPairs** in order for the fact to be converted correctly. If a fact has a regular verb and one or more words before the verb ends with an *s*, *e.g.*, glass breaks easily, then the verb in that fact must be added to **VerbPairs**.

The INTERLISP-D environment is quite different from running Franz under UNIX. Instead of creating a file like **animals.l**, you must write a LISP function that performs the same actions. Genie is still being integrated into the LISP machine environment, so concrete details cannot be given at this time.

XI. FUTURE ENHANCEMENTS

The user cannot change his answers (except in the error-checking sense of Appendix G) in the current versions of Genie; if he makes a mistake he must start over from the beginning. An algorithm has been developed (but not implemented) that will allow the user to "undo" a series of inferences and restore the knowledge base to an earlier state. He can then give a different answer and continue with the analysis from that point. The rule writer could use this feature to play "what if?" games to test a new KB.

An expert system currently under development will use multiple knowledge bases, where the first KB performs a high-level identification and the other KBs refine the analysis and arrive at an exact answer. If all of the rules were combined into a single KB it would be too large and complex to test thoroughly. Genie will be modified so that each conclusion will tell the engine what other KBs need to be run; all the information learned in one sub-analysis will be saved so the next KB can access it. This approach has the same advantages of modular programming, *e.g.*, each KB may be tested and then plugged into the entire system and modules may be used in more than one system. A multiple KB system acts like a group of specialists, each working on a different part of the problem.

LISP machines provide many features that are not available in traditional operating systems like UNIX. The Franz implementation of Genie was carefully kept machine and terminal independent and has been installed with little or no modifications at several sites. On the other hand, this has weakened the user interface. The INTERLISP-D version of Genie uses multiple windows, pop-up menus, and graphics, and the user may communicate with Genie by selecting items from a menu with the mouse or by typing his commands like he does with the Franz version (the INTERLISP-D implementation is a superset).

Most of the extensions to Genie have been for the user's benefit. Powerful tools need to be developed to assist the rule writer as well. An interactive, mouse- and window-based rule editor would be a major improvement to the way rules must be written now. The rule writer would not have to worry about syntax or other mundane matters and could concentrate on writing good rules. Allowing him to browse through the knowledge base and observe its structure would shorten the debugging process and clarify what he is attempting to do.

Genie will continue to evolve as more people write knowledge bases and make suggestions for new Genie enhancements. These changes will be prioritized on the basis of their usefulness and made as time permits.

APPENDIX A

Knowledge-Base Keywords

IF:

```
usage:          define an if/then rule
form:           (IF FactStmt [(Weight)] FactStmt [(Weight)] ...
                 [NEED (Number)]
                 THEN FactStmt FactStmt ...)
restrictions:   rules may not be logically recursive
                facts in conclusion may not be numeric
examples:
                (IF (detail level is intelligence information)
                     (number of stages GT 8)
                 THEN (intelligence info indicates axial compressor))
                (IF (compressor has driver rings)
                     (there are vane lever arms)
                 NEED (1))
                 THEN (compressor has variable geometry))
                (IF (animal is ungulate (10))
                     (markings are dark spots (10))
                     (animal has long neck (1))
                     (animal has long legs (1))
                 NEED (21)
                 THEN (animal is giraffe))
```

XOR

```
usage:          declare facts to be mutually exclusive
                if one fact is true the rest will be memorized as false
form:           (XOR PosFact PosFact ...)
restrictions:   facts must be affirmative
examples:       (XOR (compressor has variable geometry)
                       (compressor has fixed geometry))
                (XOR (animal is mammal) (animal is bird)
                       (animal is fish))
```

HYP

```
usage           declare a fact to be a hypothesis
form:           (HYP PosFact PosFact ...)
restrictions:   facts must be affirmative
example         (HYP (compressor resembles a T58)
                       (compressor resembles a J57))
```

QUESTION:

```
usage:          define question to be used instead of generating one
form            (QUESTION PosFact Question)
                (QUESTION AofC Question)
restrictions.   fact must be affirmative
examples:       (QUESTION (there is a diameter decrease before compressor)
                          (Is there a significant decrease in the
                             diameter before the compressor?))
                (QUESTION (number of stages)
                          (How many stages are in the compressor?))
```

MENU:

```
  usage:         define multiple choice fact menu
                 facts will be made mutually exclusive
  form:          (MENU Template [Question] (Choice Choice ...))
  restrictions:  facts must be affirmative
  examples:      (MENU (compressor housing is X)
                       ((mild steel) (aluminum)))
                 (MENU (detail level is X) (What is the level of detail?)
                       ((low) (medium) (high)))
```

ASK-FIRST:

```
  usage:         force Genie to verify (ask) these facts or numbers first
  form:          (ASK-FIRST FactStmt FactStmt ... AofC AofC ...)
  restrictions:  none
  example:       (ASK-FIRST (compressor has variable geometry)
                            (number of stages)
                            (detail level is high))
```

DEFAULT:

```
  usage:         declare truth to be assumed if fact cannot be proven
  form           (DEFAULT (PosFact FactStmt) (PosFact FactStmt) ...)
  restrictions:  none, but FactStmt will usually be negative
  example:       (DEFAULT ((compressor has variable geometry)
                           (compressor does not have variable geometry)))
```

NUMBER:

```
  usage:         define parameters for numeric variable
                 UNITS      = units associated with variable
                 EQN        = form to evaluate if user doesn't supply value
                 PRECISION  = number of decimal points to use when
                                displaying value
                 RANGE      = bounds for checking validity of user input
  form           (NUMBER  AofC
                          [(UNITS DataUnits)]
                          [(EQN Formula)]
                          [(PRECISION Integer)]
                          [(RANGE Number OrderRelat X OrderRelat Number)]
                          [(RANGE X Relat Number)])
  restrictions:  none
  examples:      (NUMBER  (speed of shaft)
                          (UNITS rpm)
                          (EQN (plus 72788 (times
                                    (log (airflow of system)) -1.0873)))
                          (PRECISION 1)
                          (RANGE X GT 0.0))
                 (NUMBER  (time of day)
                          (UNITS minutes EST)
                          (RANGE 0 LT X LE 2400))
```

NOSHOW:

usage:	do not tell user when this fact is deduced
form:	(**NOSHOW** PosFact PosFact ...)
restrictions:	facts must be affirmative
example:	(**NOSHOW** (T58 inlet-diameter) (T58 compression-ratio))

NULL-HYP:

usage:	define statement to be displayed if no solution found
form:	(**NULL-HYP** Statement)
restrictions:	none
example:	(**NULL-HYP** (Data does not match any known compressor.))

APPENDIX B

Glossary of Terms

FactStmt	= list of English words with no punctuation there are 3 forms -- "ordinary" statements made of words: (compressor housing is made of steel) (turbine does not have driver rings) simple arithmetic statements: (Formula Relat Formula) (number of stages = = 6) ((quotient (length of rod) (diameter of rod)) GT 2.0) bounded arithmetic statements: (Number OrderRelat Formula OrderRelat Number) (15.5 LE diameter of inlet LE 18.5)
PosFact	= affirmative FactStmt (engine has variable geometry)
Formula	= numeric constant, attribute of concept, or LISP expression with AofC for each variable 3.14 number of stages –or– (number of stages) (times (width of box) (height of box))
Eqn	= another word for Formula; used to generate value for variable that user didn't know
Relat	= numerical relational operator LT or < LE or < = EQ or = or = = NE or < > or != GE or > = GT or >
OrderRelat	= one of two ordered relational operators LT or < or LE or < =
Number	= any numeric constant
AofC	= specification of a Concept and one of its Attributes (number of stages) (thrust of engine)
Concept	= name [atom] of frame of fundamental "thing" stages engine
Attr	= attribute [atom] slot of Concept number thrust
FactCode	= code symbol which names a fact frame fact8 fact142
RuleCode	= code symbol which names a rule frame rule4 rule63
MenuCode	= code symbol which names a menu frame menu2 menu5

XorCode	= code symbol which contains a list of FactCodes
	xor3 ← (fact8 fact12)
	xor17 ← (fact5 fact37 fact25)

HypCode	= code for fact which is a hypothesis

FactPair	= list of FactCode and Sense (fact8 1) (fact27 0)
FactTriplet	= list of FactCode, Sense, and Weight (fact8 1 6) (fact27 0 1)
Sense	= fact is either affirmative [1] or negative [0]
State	= truth of a fact or status of a rule 0, 1, -1
0	= fact: negative; rule: failed
1	= fact: affirmative; rule: fired
-1	= indeterminate
Weight	= number [usually an integer] appended to a FactStmt in the **IF** portion of a **NEED** rule. *there are not weights in the classic sense*, but are used to implement a logical *OR* in an **IF/THEN** rule.
Question	= list of English words in the form of a question (Can you see any vanes?) (What is the rated thrust of the engine?)
Template	= base portion of multiple choice fact (compressor housing is made of **X**) (**X** brackets support the pump) (turbine has **X** vanes)
Choice	= list to be substituted for **X** in Template to make FactStmt (cold rolled steel) (aluminum)
Fragment	= another word for Choice
Reason	= *how a fact or numeric* concept was learned deduced given calculated
DataUnits	= units that number is in, displayed with prompt units are not "understood" by program feet pounds per square inch

APPENDIX C

Animal Knowledge Base

```
; This is the Animals data as used by Genie version 5.5

; = = = = = = =  If/Then rules  = = = = = = =

; determine general class [mammal, bird, fish]

(IF (animal has hair)
    (number of legs LE 4)
 THEN (animal is mammal))
(IF (animal gives milk)
 THEN (animal is mammal))
(IF (animal has feathers)
 THEN (animal is bird))
(IF (animal flies)
    (animal lays eggs)
    (number of legs EQ 2)
 THEN (animal is bird))
(IF (animal has scales)
    (animal has fins)
    (animal swims)
 THEN (animal is fish))

; determine subclass [carnivore, ungulate, feline]

(IF (animal eats mostly meat)
 THEN (animal is carnivore))
(IF (animal has pointed teeth)
    (animal has claws)
    (animal has forward-pointing eyes)
 NEED (1)
 THEN (animal is carnivore))
(IF (animal is mammal)
    (animal has hoofs)
 THEN (animal is ungulate))
(IF (animal is mammal)
    (animal chews cud)
 THEN (animal is ungulate)
      (animal is even toed))
(IF (animal is mammal)
    (animal is carnivore)
    (animal has tawny color)
 THEN (animal is feline))

; determine specific mammals

(IF (animal is mammal)
    (animal flies)
 THEN (animal is bat))
(IF (animal is feline)
    (markings are dark spots)
 THEN (animal is cheetah))
```

```
(IF (animal is feline)
    (markings are black stripes)
 THEN (animal is tiger))
(IF (animal is feline)
    (markings are plain)
    (animal roars)
 THEN (animal is lion))
(IF (animal is ungulate (10))
    (markings are dark spots (10))
    (animal has long neck (1))
    (animal has long legs (1))
 NEED (21)
 THEN (animal is giraffe))
(IF (animal is ungulate)
    (markings are black stripes)
 THEN (animal is zebra))

; determine specific birds

(IF (animal is bird (10))
    (animal does not fly (10))
    (animal is black and white (10))
    (animal has long neck (1))
    (animal has long legs (1))
 NEED (31)
 THEN (animal is ostrich))
(IF (animal is bird)
    (animal does not fly)
    (animal is black and white)
    (animal swims)
 THEN (animal is penguin))
(IF (animal is bird)
    (animal has red breast)
 THEN (animal is robin))
(IF (animal is bird)
    (animal is carnivore)
    (animal hunts at night)
 THEN (animal is owl))
(IF (animal is bird)
    (animal is carnivore)
    (animal is scavenger)
 THEN (animal is vulture))

; determine specific fishes

(IF (animal is fish)
    (animal is bought at pet stores)
 THEN (animal is goldfish))
```

```
(IF (animal is fish)
    (animal is carnivore)
    (animal is big)
 THEN (animal is shark))
(IF (animal is fish)
    (animal is carnivore)
    (animal fits into glass bowl)
 THEN (animal is piranha))

; * * * * * * *  Hypotheses  * * * * * * *

; mammals

(HYP (animal is bat))
(HYP (animal is tiger) (animal is cheetah) (animal is lion))
(HYP (animal is giraffe) (animal is zebra))

; birds

(HYP (animal is ostrich) (animal is penguin))
(HYP (animal is robin))
(HYP (animal is owl) (animal is vulture))

; fishes

(HYP (animal is goldfish))
(HYP (animal is shark) (animal is piranha))

; * * * * * * *  Mutually-exclusive Facts  * * * * * * *

(XOR (animal is mammal) (animal is bird) (animal is fish))

; * * * * * * *  Custom Questions  * * * * * * *

(QUESTION (animal is scavenger) (Is the animal a scavenger?))

; * * * * * * *  Menus  * * * * * * *

(MENU (animal has X) (What is the skin covering?)
      ((hair) (feathers) (scales)))
(MENU (markings are X) (What kind of markings does animal have?)
      ((black stripes) (dark spots) (plain)))

; * * * * * * *  Defaults  * * * * * * *

(DEFAULT ((animal is mammal) (animal is not mammal)))
(DEFAULT ((animal is bird) (animal is not bird)))
(DEFAULT ((animal is fish) (animal is not fish)))
(DEFAULT ((animal is carnivore) (animal is not carnivore)))
(DEFAULT ((animal is ungulate) (animal is not ungulate)))

; * * * * * * *  Default Conclusion  * * * * * * *

(NULL-HYP (You should visit the zoo more often.))
```

APPENDIX D

Animal Bootstrap File

```
; This file loads and executes the animal identification ES

; Display banner
(msg N "Animal Identification Demo" N N)

; Load inference engine
(load 'genie)

; Define specifics for this knowledge base
(setq InputFile    'animals.english)
(setq DigestedFile 'animals.frames)
(setq VerbPairs (cons '(flies fly) VerbPairs))

; Do it
(initialize)
(driver go)
```

APPENDIX E

Sample Run of

Animal Identification Expert System

```
Animal Identification Demo

Please wait whilst Genie version 5.5 is loaded.
Please wait whilst Genie loads the data.

24 rule frames loaded.
51 fact frames loaded.
2 menu frames loaded.
1 concept frames loaded.
3 exclusive-or sets loaded.
14 potential hypotheses loaded.
0 'ask-first' facts loaded.
Fact symbol tree loaded.

All data has been loaded from animals.frames.

Hello, fferd.

What is the skin covering?

   1 hair
   2 feathers
   3 scales

Enter the number of your choice-- why

Because this fact is used in determining if
  [fact13]  : animal is fish

What is the skin covering?

   1 hair
   2 feathers
   3 scales

Enter the number of your choice-- 2

Rule3 deduces animal is bird.
Because animal is bird Genie deduces animal is not mammal.
Because animal is bird Genie deduces animal is not fish.
Does animal have red breast? show rule3

IF   [fact5]   animal has feathers [succeeded]
THEN [fact6]   animal is bird [succeeded]

Does animal have red breast? find mammal

[fact3]   : animal is mammal
```

```
Does animal have red breast? how fact3

Genie learned that animal is not mammal
 because it was mutually exclusive with
   [fact6]   : animal is bird

Does animal have red breast? show xor1

The following facts are mutually exclusive
   [fact3]   : animal is mammal
   [fact6]   : animal is bird
   [fact13]  : animal is fish

Does animal have red breast? no
Does animal fly? why

Because this fact is used in determining if
  [fact38]  : animal is ostrich

Does animal fly? no
Is animal black and white? hyp

The current hypothesis is
  [fact38]  : animal is ostrich

Is animal black and white? yes
Does animal have long neck? no
Does animal have long legs? no
Does animal swim? facts

  given        : [fact5]   : animal has feathers
  deduced      : [fact6]   : animal is bird
  excluded     : [fact3]   : animal is not mammal
  excluded     : [fact13]  : animal is not fish
  excluded     : [fact10]  : animal does not have scales
  excluded     : [fact1]   : animal does not have hair
  elim (hyp)   : [fact51]  : animal is not piranha
  elim (hyp)   : [fact49]  : animal is not shark
  indeterminate: [fact24]  : animal is feline
  elim (hyp)   : [fact29]  : animal is not tiger
  given        : [fact40]  : animal does not have red breast
  elim (hyp)   : [fact41]  : animal is not robin
  given        : [fact7]   : animal does not fly
  given        : [fact37]  : animal is black and white
  given        : [fact33]  : animal does not have long neck
  given        : [fact34]  : animal does not have long legs
  elim (hyp)   : [fact38]  : animal is not ostrich
```

```
Does animal swim? yes
Rule18 deduces animal is penguin.

Final conclusion: [fact39] : Animal is penguin.

Give me a command-- how fact39

Genie deduced the hypothesis animal is penguin
 because rule18 fired.  It says

IF    [fact6]   animal is bird [succeeded]
 and  [fact7]   animal does not fly [succeeded]
 and  [fact37]  animal is black and white [succeeded]
 and  [fact12]  animal swims [succeeded]
THEN  [fact39]  animal is penguin [succeeded]

Give me a command-- why fact7

Genie needed to know if
   [fact7]   : animal flies
because it was trying to determine if
   [fact38]  : animal is ostrich
which was the current hypothesis.

Give me a command-- why fact6

Genie never tried to learn if
   [fact6]   : animal is bird

Give me a command-- why fact38

The hypothesis was
   [fact38]  : animal is ostrich

Give me a command-- end
Goodbye, fferd.
```

APPENDIX F

Numeric Test Knowledge Base

```
; This is the numeric test data as used by Genie version 5.5

; = = = = = = = =  If/Then rules  = = = = = = =

(IF (size of vmem > (times 0.8125 (capacity of vmem)))
 THEN (it is time to recopy vmem))

(IF (it is time to recopy vmem)
    (vmem update is not successful)
 THEN (you can call xerox support))

(IF (201 GT value of display)
    ((times 2 3) <= number of reboot-tries)
 THEN (you can call xerox support))

(IF (value of display EQ 1109)
    (900 LE time of day < 1100)
 THEN (you can keep working))

; = = = = = = = =  Hypotheses  = = = = = = =

(HYP (you can call xerox support) (you can keep working))

; = = = = = = = =  Custom Questions  = = = = = = = =

(QUESTION (number of reboot-tries)
          (How many times did you try to reboot?))
; special chars must be protected in Franz version
(QUESTION (number of day) (What is today\'s date?))

; = = = = = = = =  Ask First  = = = = = = =

(ASK-FIRST (number of day))

; = = = = = = = =  Numeric Info  = = = = = = =

(NUMBER (size of vmem)
        (UNITS megabytes)
        (EQN (plus (number of day) (number of reboot-tries)))
        (RANGE x GE 4)
        (PRECISION 1))

(NUMBER (capacity of vmem)
        (PRECISION 3))

(NUMBER (time of day)
        (UNITS HHmm EST)
        (EQN 1030)
        (RANGE 0 < xxxxx <= 2400))

(NUMBER (number of reboot-tries)
        (EQN (mod (time of day) 100)))

; = = = = = = = =  Default Conclusion  = = = = = = =

(NULL-HYP (You can do whatever you want.))
```

APPENDIX G

Sample Run of Numeric Expert System

```
Numeric Test Demo

Please wait whilst Genie version 5.5 is loaded.
Please wait whilst Genie loads the data.

4 rule frames loaded.
9 fact frames loaded.
0 menu frames loaded.
4 concept frames loaded.
0 exclusive-or sets loaded.
2 potential hypotheses loaded.
1 'ask-first' facts loaded.
Fact symbol tree loaded.

All data has been loaded from numtest.frames.

Hello, fferd.

What is today's date? why

Because I was told to ask this question first.

What is today's date? 30
What is the size of vmem? (in megabytes) unknown
How many times did you try to reboot? why

Because this fact is part of an equation to calculate
   size of vmem

How many times did you try to reboot? unknown
What is the time of day? (in HHmm EST) rule

The equation being evaluated is
   number of reboot-tries ← (mod (time of day) 100)

What is the time of day? (in HHmm EST) 2712
Expected 0 < time of day <= 2400
  but you entered: 2712
Do you want to change your input? yes
What is the time of day? (in HHmm EST) 904
Calculated: number of reboot-tries = 4
Calculated: size of vmem = 34
What is the capacity of vmem? 40
Rule1 deduces it is time to recopy vmem.
Is vmem update successful? yes
What is the value of display? 1109
Rule4 deduces you can keep working.

Final conclusion: [fact9] : You can keep working.
```

```
Give me a command-- facts

  deduced      : [fact2]   : it is time to recopy vmem
  given        : [fact3]   : vmem update is successful
  elim (hyp)   : [fact4]   : you can not call xerox support
  deduced      : [fact9]   : you can keep working

Give me a command-- numbers

     calculated : size of vmem = 34 megabytes
          given : capacity of vmem = 40
          given : value of display = 1109
     calculated : number of reboot-tries = 4
          given : time of day = 904 HHmm EST
          given : number of day = 30

Give me a command-- numfacts

 num-relation : [fact8] : 900 LE time of day < 1100
 num-relation : [fact6] : (times 2 3) *NOT* <= number of reboot-tries
 num-relation : [fact1] : size of vmem > (times 0.8125 (capacity of vmem))
 num-relation : [fact5] : 201 *NOT* GT value of display
 num-relation : [fact7] : value of display EQ 1109

Give me a command-- end
Goodbye, fferd.
```

DISTRIBUTION LIST

No. of Copies	Organization
2	Administrator Defense Technical Info Center ATTN: DTIC-DDA Cameron Station Alexandria, VA 22314-6145
1	Director Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209
1	HQDA ATTN: DAMA-ART-M Washington, DC 20310
1	HQDA US Army Artificial Intelligence Center ATTN: DACS-DMA, Dr. A. Ressler Washington, DC 20310-0200
1	Commander US Army Materiel Command ATTN: AMCDRA-ST 5001 Eisenhower Avenue Alexandria, VA 22333-0001
1	Commander Armament Research & Development Center USAAMCCOM ATTN: SMCAR-TDC Dover, NJ 07801
2	Commander Armament Research & Development Center USAAMCCOM ATTN: SMCAR-TSS Dover, NJ 07801
1	Commander US Army Armament Munitions & Chemical Command ATTN: SMCAR-ESP-L Rock Island, IL 61299
1	Director Benet Weapons Laboratory Armament Research & Development Center USAAMCCOM ATTN: SMCAR-LCB-TL Watervliet, NY 12189
1	Commander US Army Aviation Research & Development Command ATTN: AMSAV-E 4300 Goodfellow Blvd St. Louis, MO 63120
1	Commander Army Aviation System Command ATTN: AMSAV-DSSB, N. Bodenstein 4300 Goodfellow Blvd St. Louis, MO 63120-1798
1	Director US Army Air Mobility Research & Development Laboratory Ames Research Center Moffett Field, CA 94035
1	Director Appl. Tech. Directorate USAARTA (AVSCOM) ATTN: DAVDL-EU-SY-RPV Fort Eustis, VA 23604
1	Commander US Army Communications- Electronics Command ATTN: AMSEL-ED Fort Monmouth, NJ 07703
1	Commander US Army Communications- Electronics Command ATTN: AMSEL-RD-COM-IR-1, Dr Powell Fort Monmouth, NJ 07703-5204

DISTRIBUTION LIST

No. of Copies	Organization
1	Commander ERADCOM Technical Library ATTN: DELSD-L (Reports Section) Fort Monmouth, NJ 07703-5301
1	Commander US Army Missile Command Research, Development, & Engineering Center ATTN: AMSMI-RD Redstone Arsenal, AL 35898
1	Director US Army Missile & Space Intelligence Center ATTN: AIAMS-YDL Redstone Arsenal, AL 35898-5500
1	Commander US Army Tank Automotive Command ATTN: AMSTA-TSL Warren, MI 48397-5000
2	Commander US Army Research Office ATTN: AMXRO-MA, Dr. J. Chandra Dr. R. Green P.O. Box 12211 Research Triangle Park, NC 27709-2211
1	Commander US Army Combat Dev & Experimentation Command ATTN: ATEC-SA, Dr. M. R. Bryson Ford Ord, CA 93941
1	Commander US Army Harry Diamond Laboratory ATTN: DELHD-RT-RD 2800 Powder Mill Rd Adelphi, MD 20783
1	Director US Army TRADOC Systems Analysis Activity ATTN: ATAA-SL, Tech Lib White Sands Missile Range, NM 88002
1	Commandant US Army Infantry School ATTN: ATSH-CD-CSO-OR Fort Benning, GA 31905
1	Commander US Army Development & Employment Agency ATTN: MODE-TED-SAB Fort Lewis, WA 98433
1	Commandant US Military Academy ATTN: Lt. Col. J. Dallen West Point, NY 10996
1	AFWL/SUL Kirtland AFB, NM 87117
1	Air Force Armament Laboratory ATTN: AFATL/DLODL Eglin AFB, FL 32542-5000
1	AFELM, The Rand Corporation ATTN: Library-D 1700 Main Street Santa Monica, CA 90406
1	Battelle Columbus Laboratories ATTN: Ordnance Div 505 King Avenue Columbus, OH 43201
2	Southwest Research Inst. Dept. of Mech. Sciences ATTN: Mr. A. Wenzel Mr. P. Zabel 8500 Culebra Road San Antonio, TX 78284

DISTRIBUTION LIST

No. of Copies	Organization
1	Oklahoma State University Field Office ATTN: Mrs. Ann Peebles P.O. Box 1925 Eglin Air Force Base, FL 32542
1	Prof. Lotfi Zadeh Dept. of Electrical Engr. & Comp. Science University of California Berkeley, CA 94720
1	Prof. James T. P. Yao School of Civil Engr. Civil Engr. Bldg. Purdue University, West Lafayette, IN 47907
1	Prof. J. L. Chameau Geotech Engr. Grissom Hall Purdue University, West Lafayette, IN 47907
1	Dr. William H. Friedman Dept. Econ. & Dec. Sci. Loyola College 4501 N. Charles St. Baltimore, MD 21210-2699
1	Machine Intelligence Inst. Iona College ATTN: Dr. R. Yager New Rochelle, NY 10801
1	Dr. Felix S. Wong Weidlinger Associates 620 Hansen Way Suite 100 Palo Alto, CA 94304
1	Dr. Wilfred E. Baker 8103 Broadway San Antonio, TX 78209
1	Dr. Joseph Sperrazza 1000 Cedar Corner Perryville, MD 21903
1	Dr. Steven B. Boswell MIT Lincoln Lab, Group 94 Lexington, MA 02173-0073
1	Robert L. Stewart Johns Hopkins University Applied Physics Laboratory Laurel, MD 20707
	Aberdeen Proving Ground
2	Dir, USAMSAA ATTN: AMXSY-D AMXSY-MP (H. Cohen)
1	Cdr, USATECOM ATTN: AMSTE-TO-F
3	Cdr, CRDC, AMCCOM, Bldg. E3516 ATTN: SMCCR-RSP-A SMCCR-MU SMCCR-SPS-IL

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. BRL Report Number ______________________ Date of Report __________

2. Date Report Received ______________________________________

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) ______________

4. How specifically, is the report being used? (Information source, design data, procedure, source of ideas, etc.) ______________

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided or efficiencies achieved, etc? If so, please elaborate. ______________

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.)

CURRENT ADDRESS	Name
	Organization
	Address
	City, State, Zip

7. If indicating a Change of Address or Address Correction, please provide the New or Correct Address in Block 6 above and the Old or Incorrect address below.

OLD ADDRESS	Name
	Organization
	Address
	City, State, Zip

(Remove this sheet, fold as indicated, staple or tape closed, and mail.)

— — — — — — — — — — — FOLD HERE — — — — — — — — — — — —

Director
US Army Ballistic Research Laboratory
ATTN: DRXBR-OD-ST
Aberdeen Proving Ground, MD 21005-5066

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, $300

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 12062 WASHINGTON,DC

POSTAGE WILL BE PAID BY DEPARTMENT OF THE ARMY

Director
US Army Ballistic Research Laboratory
ATTN: DRXBR-OD-ST
Aberdeen Proving Ground, MD 21005-9989

— — — — — — — — — — FOLD HERE — — — — — — — — — — — —

END
12-87
DTIC